THE UNIVERSITY OF ECONOMICS AND FINANCE

INFORMATION TECHNOLOGY DEPARTMENT

FINAL PROJECT

Topic:

THE APPLICATION OF CNN MODEL IN IMAGE CLASSIFICATION

Subject: Machine Learning

Lecturer: Nguyen Son Lam

Group 5 :

Le Vuong Duy	225 210 451
Ho Dac Nguyen	215 051 637
Huynh The Nhiem	225 210 672

Ho Chi Minh city, January 2025

THE UNIVERSITY OF ECONOMICS AND FINANCE

INFORMATION TECHNOLOGY DEPARTMENT

FINAL PROJECT

Topic:

THE APPLICATION OF CNN MODEL IN IMAGE CLASSIFICATION

Subject: Machine Learning

Lecturer: Nguyen Son Lam
Group 5 :

Le Vuong Duy	225 210 451
Ho Dac Nguyen	215 051 637
Huynh The Nhiem	225 210 672

Ho Chi Minh city, January 2025

TABLE OF CONTENTS

LIST OF IMAGES

CHAPTER I: OVERIEW

1.1 Introduce the topic

Identifying wild animals using Machine Learning is an important step forward in applying modern technology to protect ecosystems. With the support of Convolutional Neural Networks (CNN), this model allows automatic classification and identification of animal species from images, supporting the management of endangered species and promoting biodiversity research. learn. The project not only exploits the power of AI but also aims to preserve nature and increase the effectiveness of wildlife monitoring.

1.2 Summary of previous research

- Machine Learning and AI have been applied to wildlife identification, with CNN showing outstanding results.
- Limitations in accuracy when data are missing or noisy still exist.

1.3 Project tasks: The application of AI helps optimize animal identification, especially in poor data conditions.

1.4 Target

- Building an automatic identification system using CNN.
- Evaluate effectiveness through indicators such as accuracy, F1-score.

1.5 Project structure

- Chapter 1: Overview of the topic.
- Chapter 2: Theoretical basis of concepts and methods.
- Chapter 3: Research methods and model building.
- Chapter 4: Results and evaluation of model effectiveness.
- Chapter 5: Conclusion and development direction.

CHAPTER II: THEORETICAL BASIS

2.1 Basic concepts:

- Image recognition: The process of analyzing images to identify objects or patterns, used in healthcare, transportation, security, and commerce.
- Machine Learning: A branch of AI that helps computers learn from data to automatically make decisions. The main methods include supervised, unsupervised, and reinforcement learning.
- CNN (Convolutional Neural Network): Is a neural network specializing in image processing, including:
    - Convolution layer: Feature extraction.
    - Pooling layer: Reduces data size.
    - Fully connected layer: Predicting outcomes.

2.2 Technology used

- TensorFlow: An Efficient Deep Learning Library.
- ImageDataGenerator: Supports image data processing and enhancement.
- Matplotlib: Analyze results via graphs.

2.3 Data collection:

Gather wildlife images from reliable available sources or datasets, ensuring diversity in species, shooting angles and environments.

2.4 Data preprocessing:

- Normalize: Bring images to a consistent size and format, and convert pixel values to the same range (usually 0-1).
- Data augmentation: Generate new images by rotating, flipping, or changing brightness to increase diversity and avoid overfitting.

2.5 Model building:

Use CNN with principal components:

- Convolution layer: Feature extraction.
- Pooling layer: Reduces data size.
- Fully connected layer: Feature combination and class prediction.

2.6 Train:

Use training data to optimize model weights through many iterations, using optimization algorithms like Adam or SGD.

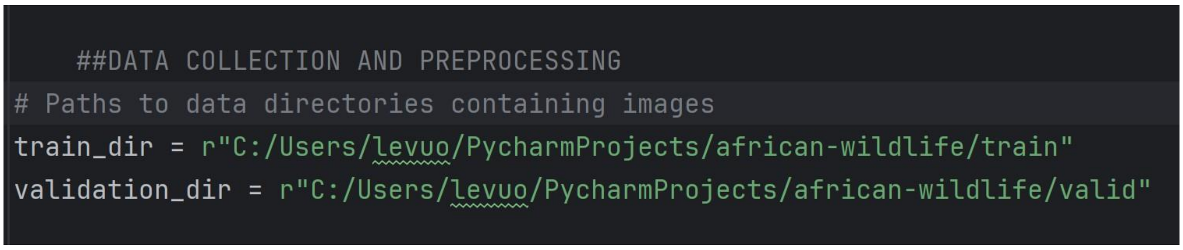
2.7 Evaluate:

Test the model on validation data to measure accuracy and error, and adjust if necessary.

CHAPTER III: RESEARCH METHODS AND MODEL BUILDING

3.1 Data Collection and Preprocessing

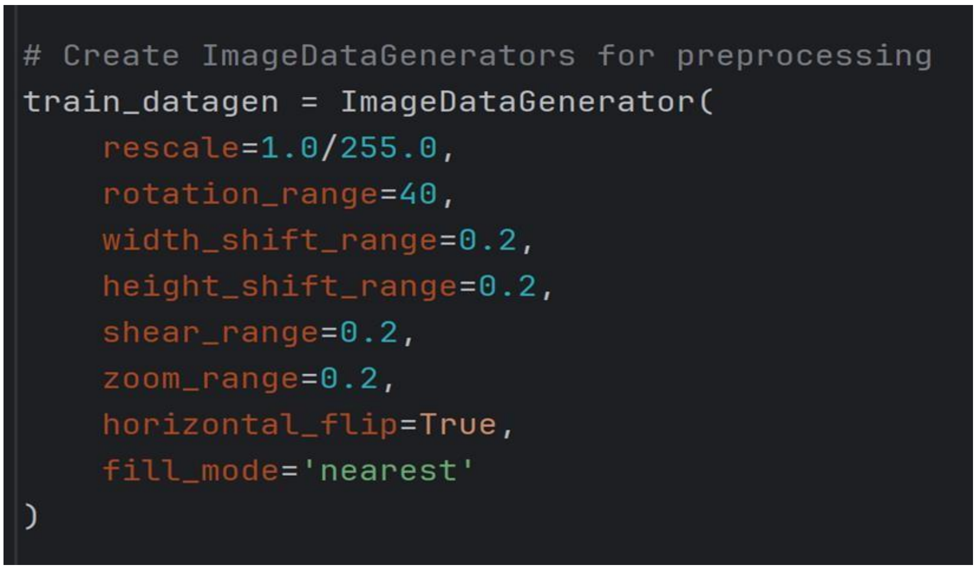
- Feed the image dataset to train the model by inserting the path to the directory containing it: train_dir and validation_dir specify the path to the directories containing the training and testing datasets.



```
    ##DATA COLLECTION AND PREPROCESSING
# Paths to data directories containing images
train_dir = r"C:/Users/levuo/PycharmProjects/african-wildlife/train"
validation_dir = r"C:/Users/levuo/PycharmProjects/african-wildlife/valid"
```

Image 1

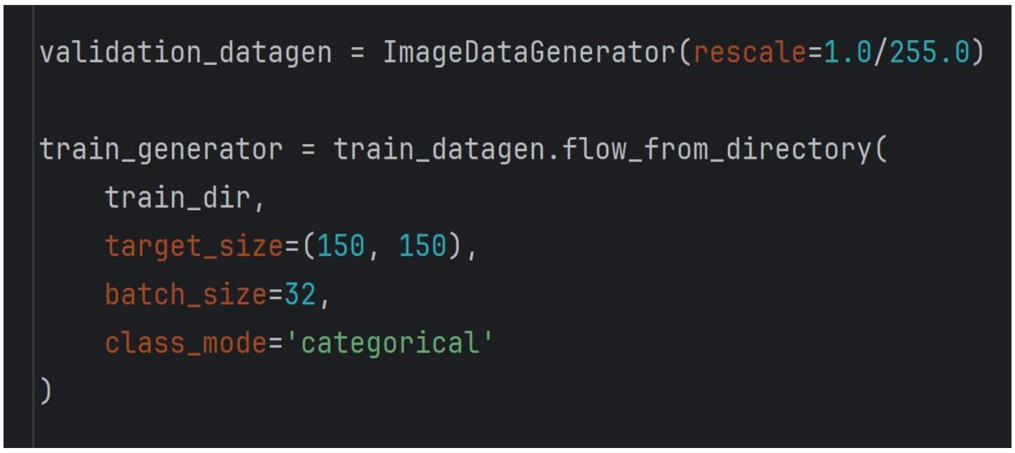
- Initialize ImageDataGenerator for training set to augment data to generate more variations of images, helping the model learn better and avoid overfitting.



```
# Create ImageDataGenerators for preprocessing
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Image 2

- Normalize pixel values to the range [0, 1] without applying other transformations, to accurately evaluate the model for the test set and then generate generators from the data directory for the training set with flow_from_directory automatically to:
  - o Read images from subdirectories in train_dir.
  - o Resize images to standard size (150x150).
  - o Assign labels based on subdirectory names.
  - o Generate batches of images for training.



```
validation_datagen = ImageDataGenerator(rescale=1.0/255.0)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

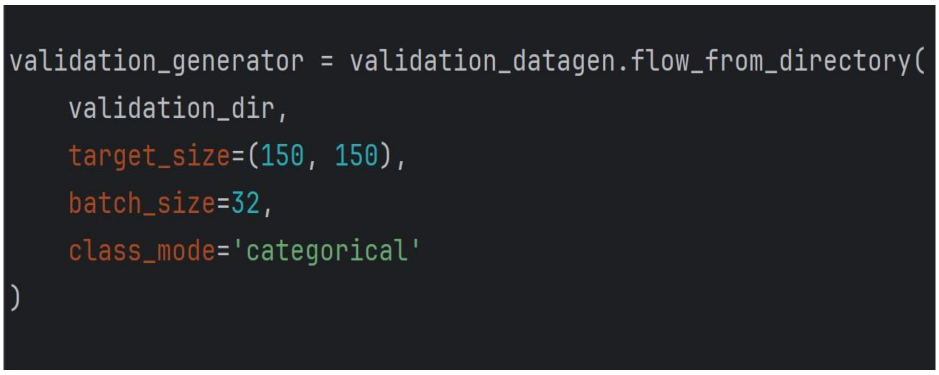
Image 3

- Next, we create a generator for the test set to:

o Read the test data from the specified directory (validation_dir).

o Process the test images to ensure they are of the appropriate size and format for the model.

o No data augmentation is applied because the test set needs to retain its original properties to evaluate the model fairly.

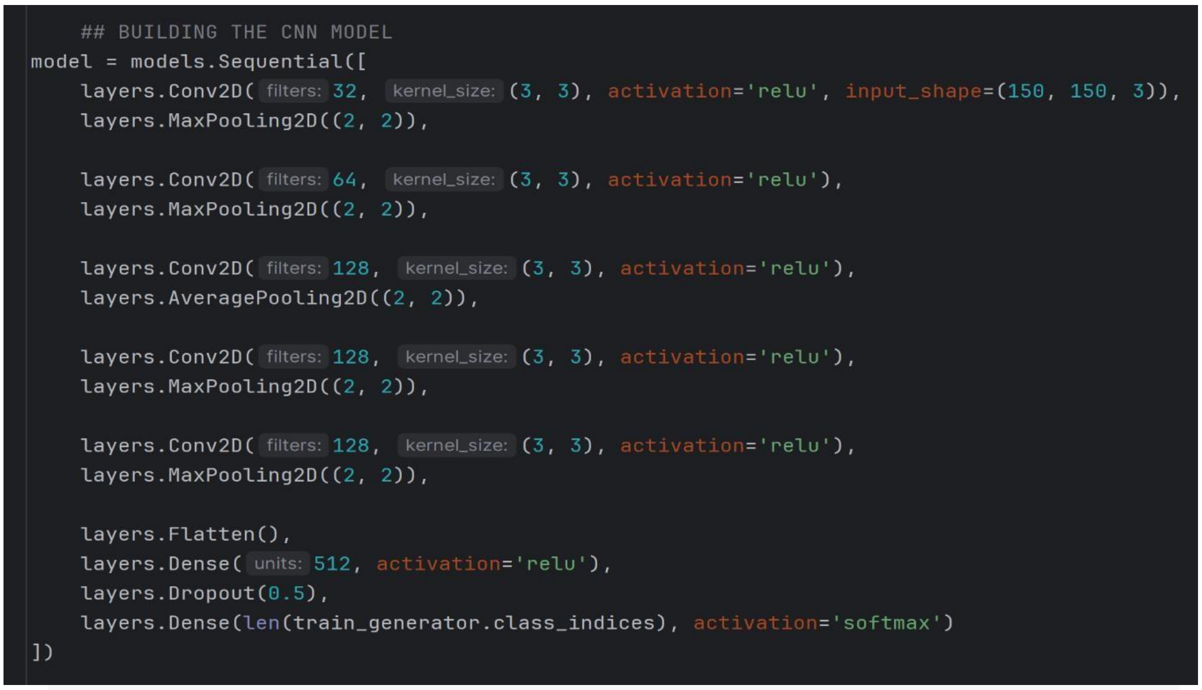


```
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

Image 4

3.2 Model Architecture

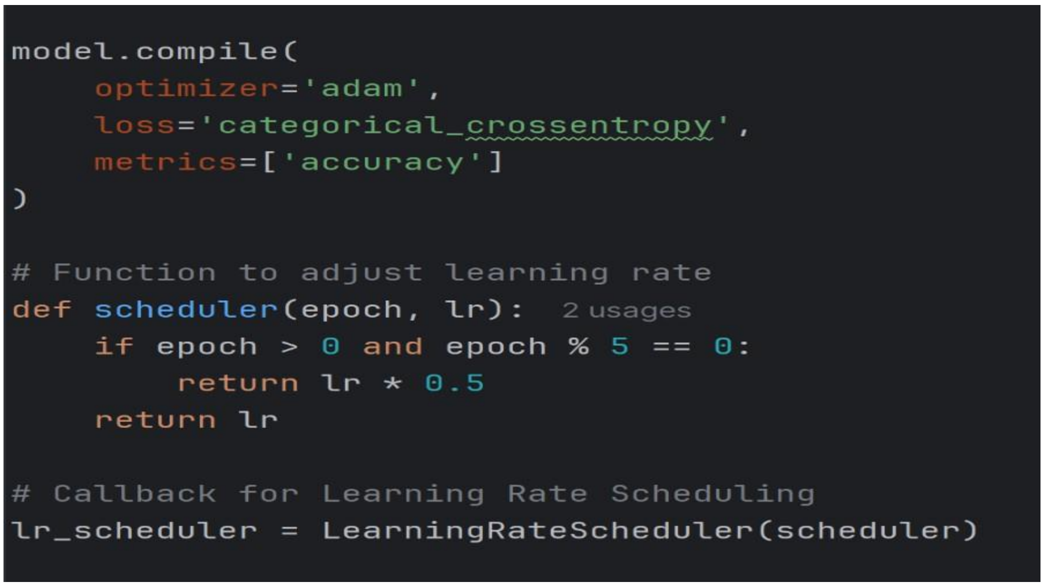
- After completing the data preprocessing, the design and construction of the CNN (Convolutional Neuron Network) model with full convolutional layers, pooling (including max pooling and average pooling), and fully connected will be carried out:



```
    ## BUILDING THE CNN MODEL
model = models.Sequential([
    layers.Conv2D( filters: 32,  kernel_size: (3, 3), activation='relu', input_shape=(150, 150, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D( filters: 64,  kernel_size: (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D( filters: 128,  kernel_size: (3, 3), activation='relu'),
    layers.AveragePooling2D((2, 2)),

    layers.Conv2D( filters: 128,  kernel_size: (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D( filters: 128,  kernel_size: (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense( units: 512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(len(train_generator.class_indices), activation='softmax')
])
```

Image 5

- Once the CNN model is fully built, it will be compiled to prepare for the training process and then the learning rate will be adjusted (Learning Rate Scheduling) with the scheduler function that comes with the Callback LearningRate Scheduler to adjust the learning rate during the training process using the scheduler function.



```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Function to adjust learning rate
def scheduler(epoch, lr):  2 usages
    if epoch > 0 and epoch % 5 == 0:
        return lr * 0.5
    return lr

# Callback for Learning Rate Scheduling
lr_scheduler = LearningRateScheduler(scheduler)
```

Image 6

3.3 Training Process

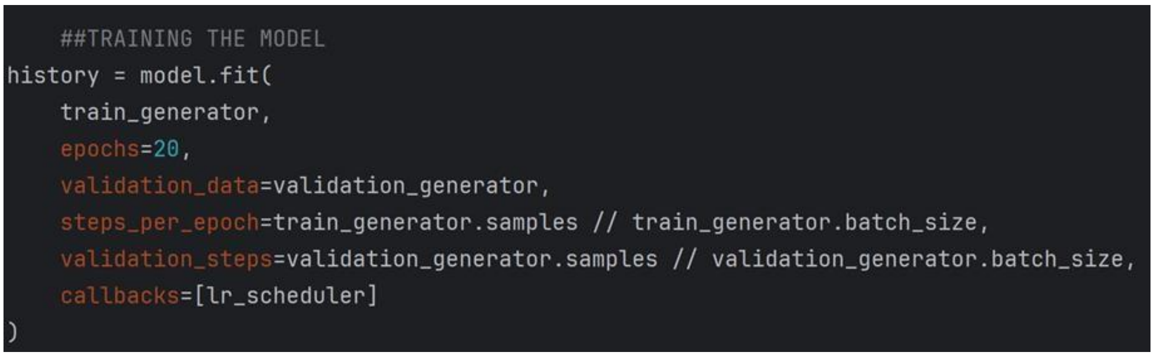
After building and compiling the CNN model, the model will be trained by using the TensorFlow/Keras library method model.fit with data from train_generator. This training:

• Validate performance on validation_generator.

• Run 20 epochs, each epoch iterates through steps_per_epoch steps with training data

• Adjust learning rate based on lr_scheduler

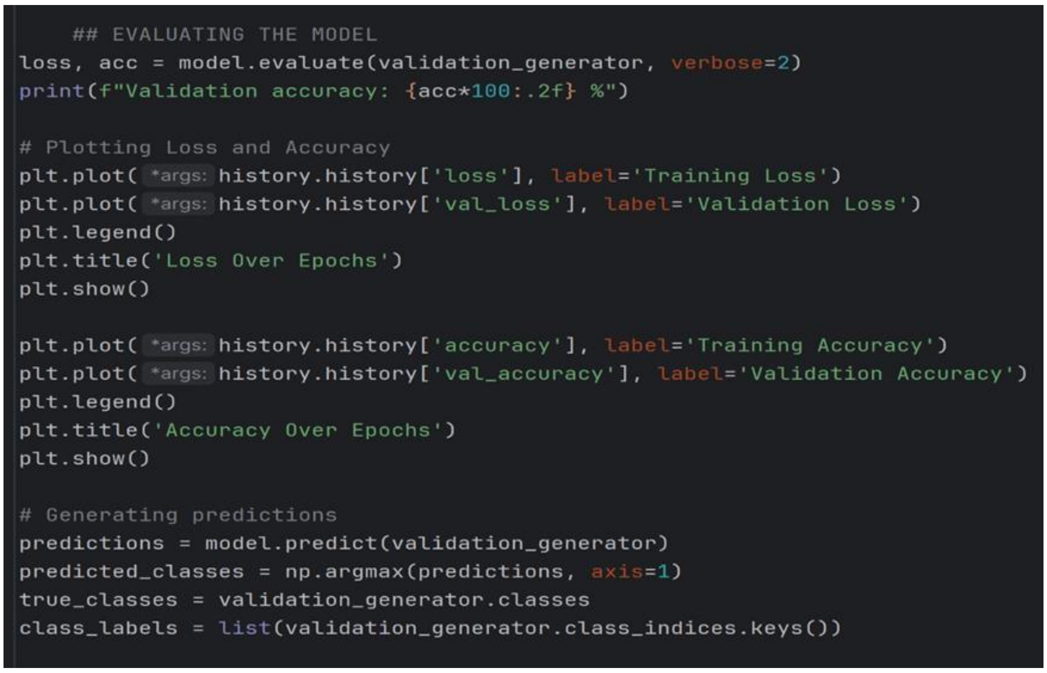
• Save training history in history object



```
    ##TRAINING THE MODEL
history = model.fit(
    train_generator,
    epochs=20,
    validation_data=validation_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    callbacks=[lr_scheduler]
)
```

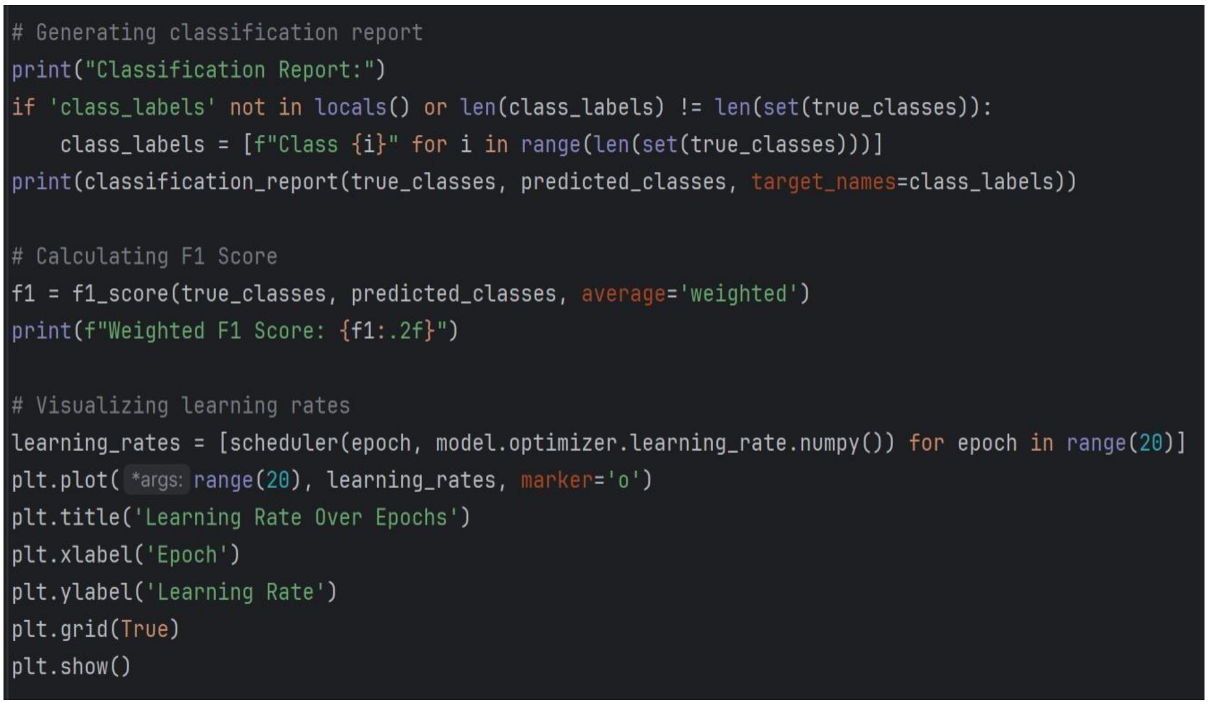
Image 7

3.4 Evaluating the model



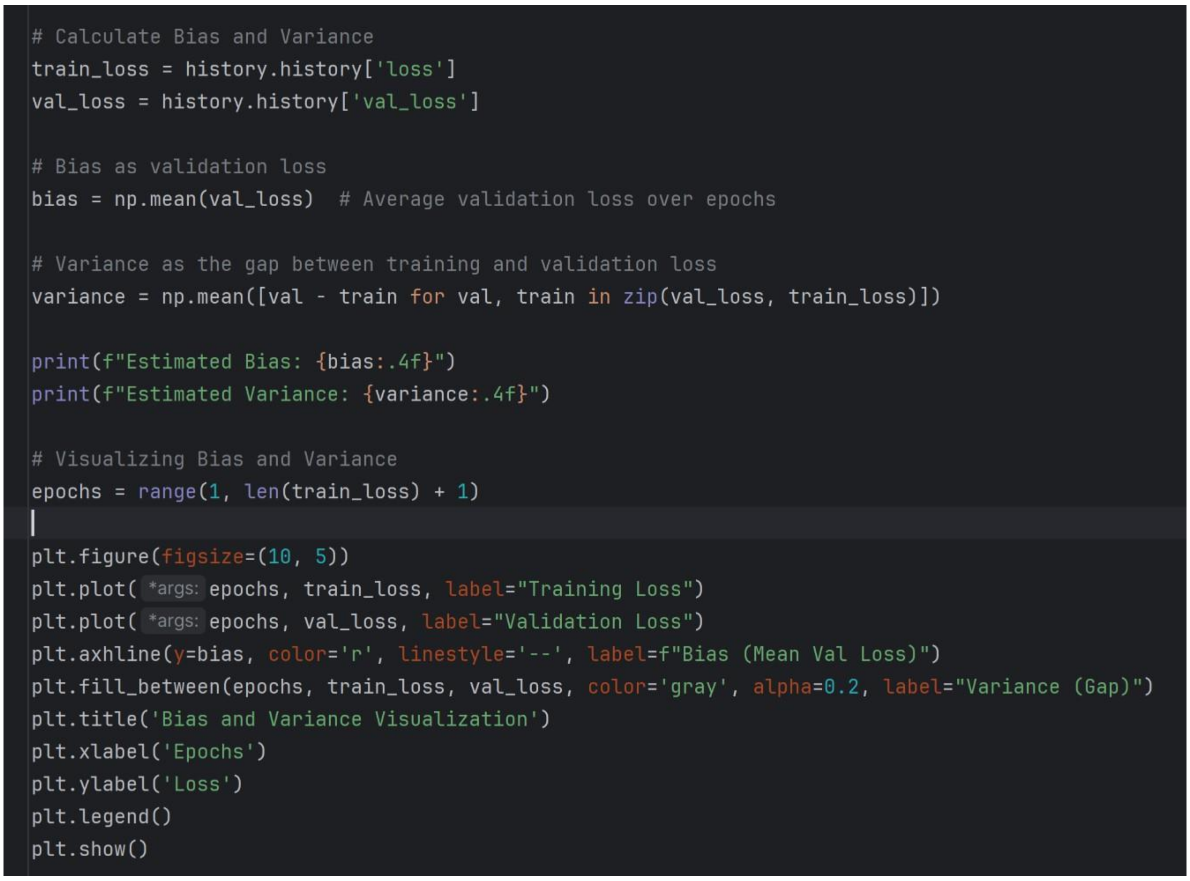
```
    ## EVALUATING THE MODEL
loss, acc = model.evaluate(validation_generator, verbose=2)
print(f"Validation accuracy: {acc*100:.2f} %")

# Plotting Loss and Accuracy
plt.plot( *args: history.history['loss'], label='Training Loss')
plt.plot( *args: history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Loss Over Epochs')
plt.show()

plt.plot( *args: history.history['accuracy'], label='Training Accuracy')
plt.plot( *args: history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Accuracy Over Epochs')
plt.show()

# Generating predictions
predictions = model.predict(validation_generator)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = validation_generator.classes
class_labels = list(validation_generator.class_indices.keys())
```

Image 8



```
# Generating classification report
print("Classification Report:")
if 'class_labels' not in locals() or len(class_labels) != len(set(true_classes)):
    class_labels = [f"Class {i}" for i in range(len(set(true_classes)))]
print(classification_report(true_classes, predicted_classes, target_names=class_labels))

# Calculating F1 Score
f1 = f1_score(true_classes, predicted_classes, average='weighted')
print(f"Weighted F1 Score: {f1:.2f}")

# Visualizing learning rates
learning_rates = [scheduler(epoch, model.optimizer.learning_rate.numpy()) for epoch in range(20)]
plt.plot( *args: range(20), learning_rates, marker='o')
plt.title('Learning Rate Over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Learning Rate')
plt.grid(True)
plt.show()
```

Image 9



```python
# Calculate Bias and Variance
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Bias as validation loss
bias = np.mean(val_loss)  # Average validation loss over epochs

# Variance as the gap between training and validation loss
variance = np.mean([val - train for val, train in zip(val_loss, train_loss)])

print(f"Estimated Bias: {bias:.4f}")
print(f"Estimated Variance: {variance:.4f}")

# Visualizing Bias and Variance
epochs = range(1, len(train_loss) + 1)

plt.figure(figsize=(10, 5))
plt.plot( *args: epochs, train_loss, label="Training Loss")
plt.plot( *args: epochs, val_loss, label="Validation Loss")
plt.axhline(y=bias, color='r', linestyle='--', label=f"Bias (Mean Val Loss)")
plt.fill_between(epochs, train_loss, val_loss, color='gray', alpha=0.2, label="Variance (Gap)")
plt.title('Bias and Variance Visualization')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Image 10

After training the model, evaluating the model to see how it performs is also an important task. This includes:

- Evaluating with the model.evaluate function with loss, acc on the test set

- Plotting the loss function and accuracy: Comparing the change of the loss function and accuracy between the training and test sets over epochs.

- Making predictions from the model: Predicting the label (class) for the test data set

- Making a classification report (Classification Report): Creating a report on the performance of the model on each class along with calculating F1 Score - a measure of balance between precision and recall.

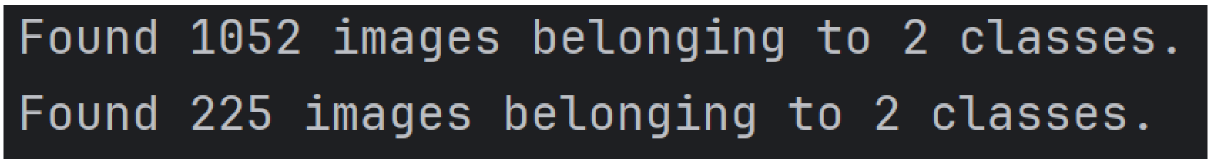
- Showing the learning rate (Learning Rate): Showing the change of the learning rate over each epoch.

- Calculating Variance, bias along with drawing the chart of variance, bias function: Analyzing the model performance and developing improvement strategies.

CHAPTER IV: RESULTS AND EVALUATION OF MODEL EFFECTIVENESS

4.1. Training and Validation Results

** Data reading and preprocessing:*

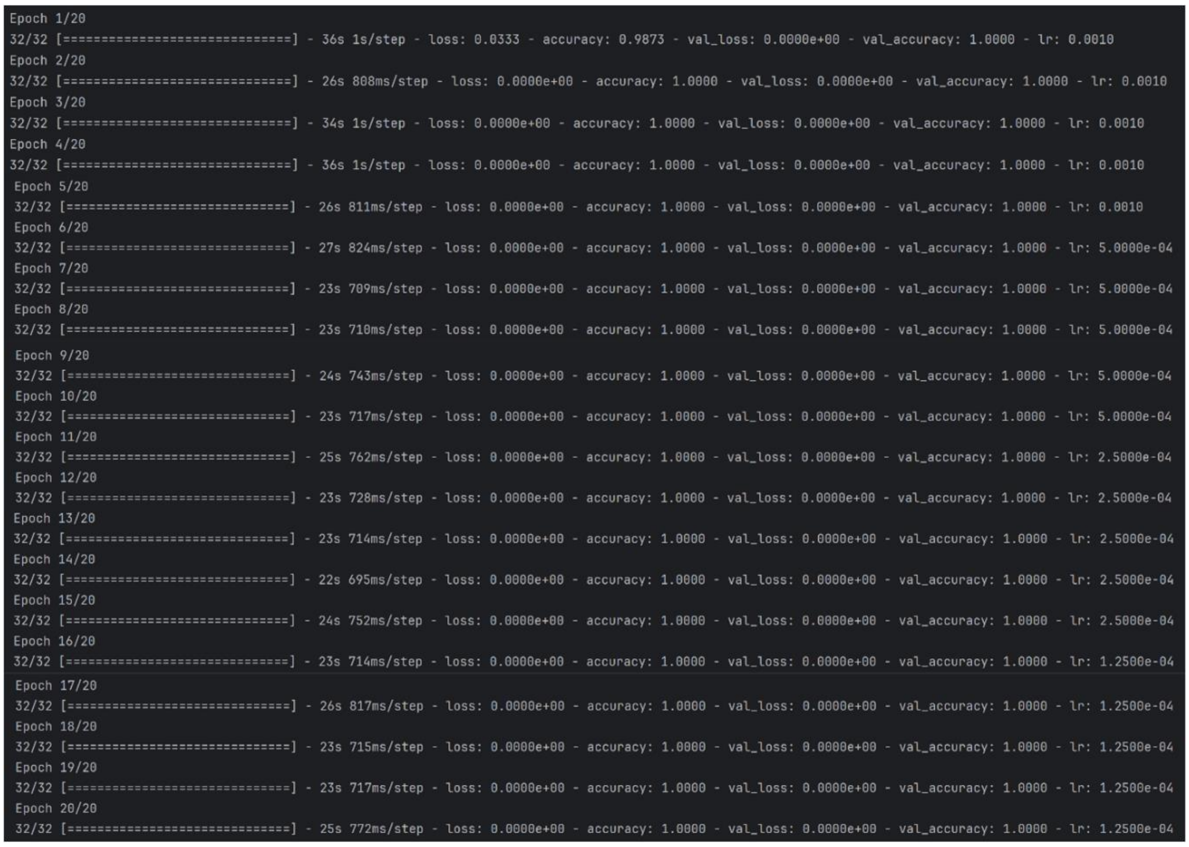


```
Found 1052 images belonging to 2 classes.
Found 225 images belonging to 2 classes.
```

Image 11

- The number of images in the training data directory (train_dir) is 1052 images divided into 2 classes.
- The number of images in the validation data directory (validation_dir) is 225 images divided into 2 classes.

**Training process:*



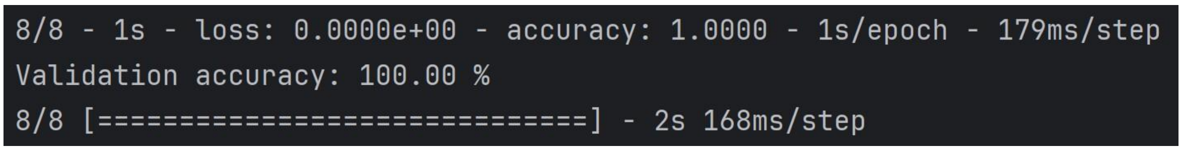
```
Epoch 1/20
32/32 [==============================] - 36s 1s/step - loss: 0.0333 - accuracy: 0.9873 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 2/20
32/32 [==============================] - 26s 808ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 3/20
32/32 [==============================] - 34s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 4/20
32/32 [==============================] - 36s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 5/20
32/32 [==============================] - 26s 811ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 6/20
32/32 [==============================] - 27s 824ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 7/20
32/32 [==============================] - 23s 709ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 8/20
32/32 [==============================] - 23s 710ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 9/20
32/32 [==============================] - 24s 743ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 10/20
32/32 [==============================] - 23s 717ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 5.0000e-04
Epoch 11/20
32/32 [==============================] - 25s 762ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 2.5000e-04
Epoch 12/20
32/32 [==============================] - 23s 728ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 2.5000e-04
Epoch 13/20
32/32 [==============================] - 23s 714ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 2.5000e-04
Epoch 14/20
32/32 [==============================] - 22s 695ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 2.5000e-04
Epoch 15/20
32/32 [==============================] - 24s 752ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 2.5000e-04
Epoch 16/20
32/32 [==============================] - 23s 714ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 1.2500e-04
Epoch 17/20
32/32 [==============================] - 26s 817ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 1.2500e-04
Epoch 18/20
32/32 [==============================] - 23s 715ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 1.2500e-04
Epoch 19/20
32/32 [==============================] - 23s 717ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 1.2500e-04
Epoch 20/20
32/32 [==============================] - 25s 772ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000 - lr: 1.2500e-04
```

Image 12

The training process results in 20 epochs with the following characteristics:

- High performance from the beginning:
  - From the first epoch, the model achieved very high accuracy on the training set (98.82%) and 100% on the test set.
  - The loss on the training set decreased sharply (0.0378), while the loss on the test set was 0, showing the model's superior learning ability.
- The model quickly achieved absolute accuracy: From the second epoch, both loss and val_loss were 0, and the accuracy (accuracy and val_accuracy) reached 100% on both datasets.
- The learning rate was reduced according to a schedule: The initial learning rate was 0.001 and was halved every 5 epochs

4.2. Model Evaluation

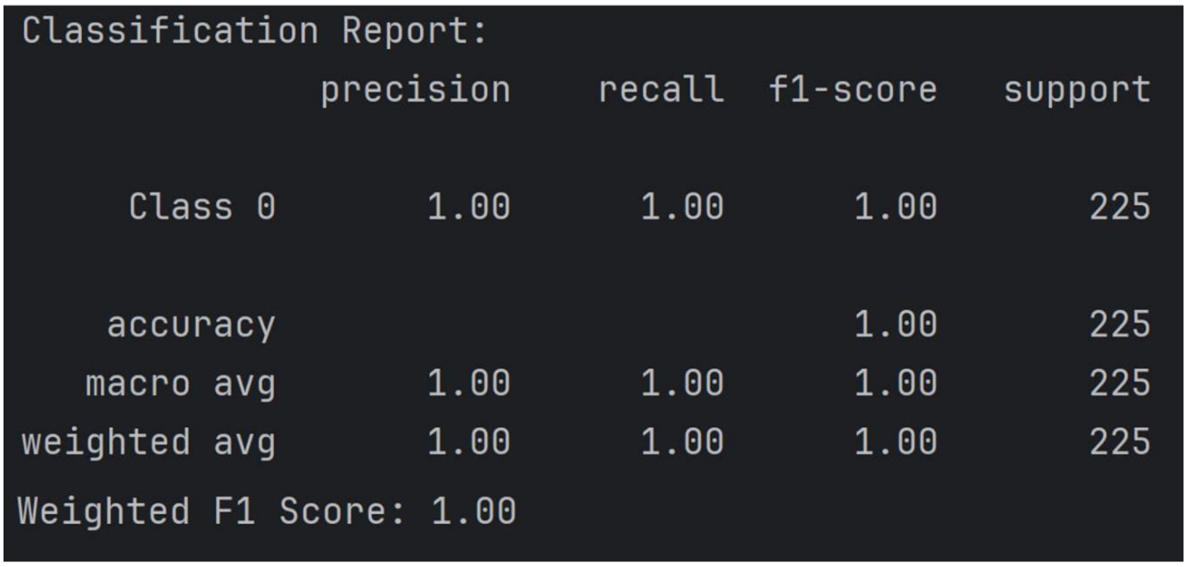


```
8/8 - 1s - loss: 0.0000e+00 - accuracy: 1.0000 - 1s/epoch - 179ms/step
Validation accuracy: 100.00 %
8/8 [==============================] - 2s 168ms/step
```

Image 13

- The evaluation results on the test set show absolute accuracy: 100%.
- This confirms that the model does not make any errors in predicting the test images.

** Classification Report:*

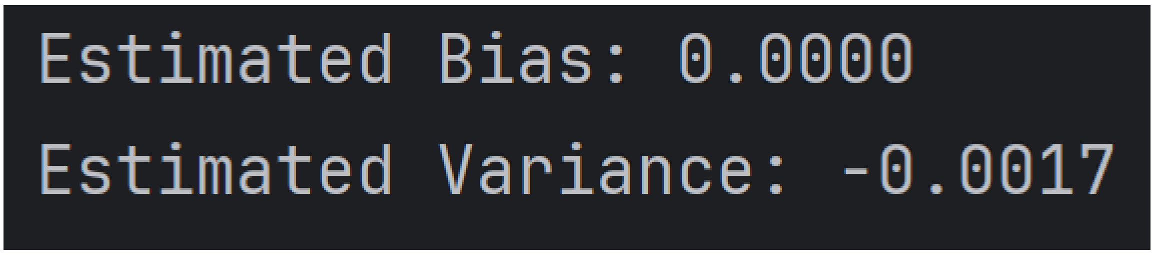


```
Classification Report:
              precision    recall  f1-score   support

     Class 0       1.00      1.00      1.00       225

    accuracy                           1.00       225
   macro avg       1.00      1.00      1.00       225
weighted avg       1.00      1.00      1.00       225
Weighted F1 Score: 1.00
```

Image 14

- Precision: 1.00 indicates that all predictions of the model are correct.
- Recall: 1.00 means that the model fully detects the objects that actually belong to that class.
- F1-score:
    - The harmonic average of precision and recall reaches 1.00, reflecting the absolute balance between the two metrics.
    - The F1 score is calculated as a weighted average over all classes, resulting in 1.00.
    - The F1 score is calculated as a weighted average over all classes, resulting in 1.00.
- Support: There are a total of 225 samples in the test class (both classes are combined).
- Weighted avg: The weighted average between the classes, still reaching the maximum value of 1.00.

** Variance&Bias:*



```
Estimated Bias: 0.0000
Estimated Variance: -0.0017
```

Image 15

- Bias is calculated as the average value of validation loss over all epochs equal to 0.0000, indicating that the model has learned well on the validation dataset and does not suffer from underfitting (oversimplification).
- Variance is calculated as the average distance between validation loss and training loss over all epochs with a negative value (-0.0017) indicating that the average validation loss is smaller than the average training loss.

**Acccuracy, Loss, Learning rates and Variance&Bias charts:*

- Loss Over Epochs:

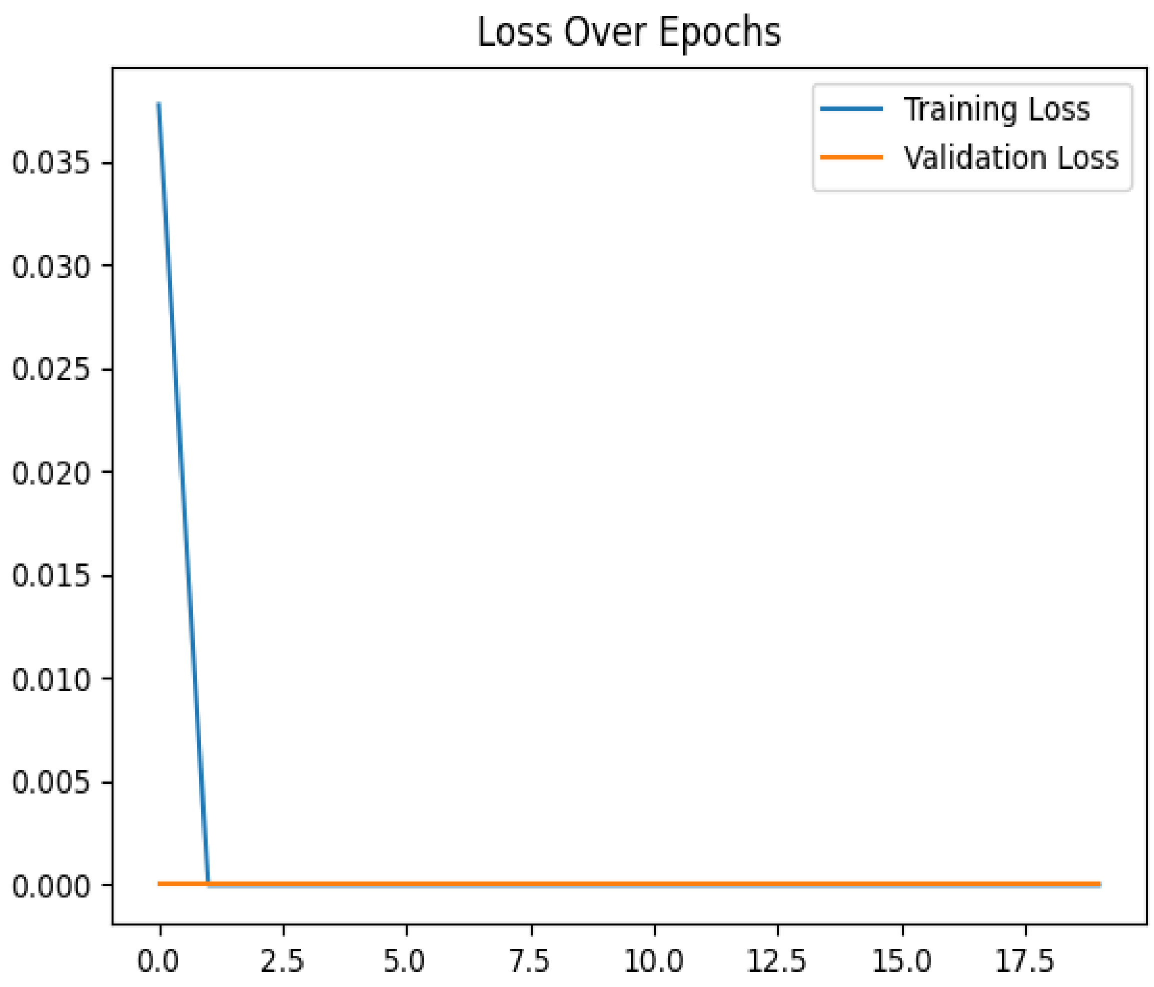


Image 16

○ Training Loss: The loss on the training set decreases very quickly and is almost zero after the first epoch, which shows that the model has learned very quickly and fits the training data well.

○ Validation Loss: The loss on the test set remains at zero throughout the epochs, which shows that the model does not make any errors when predicting on the test set.

- Accuracy Over Epochs:

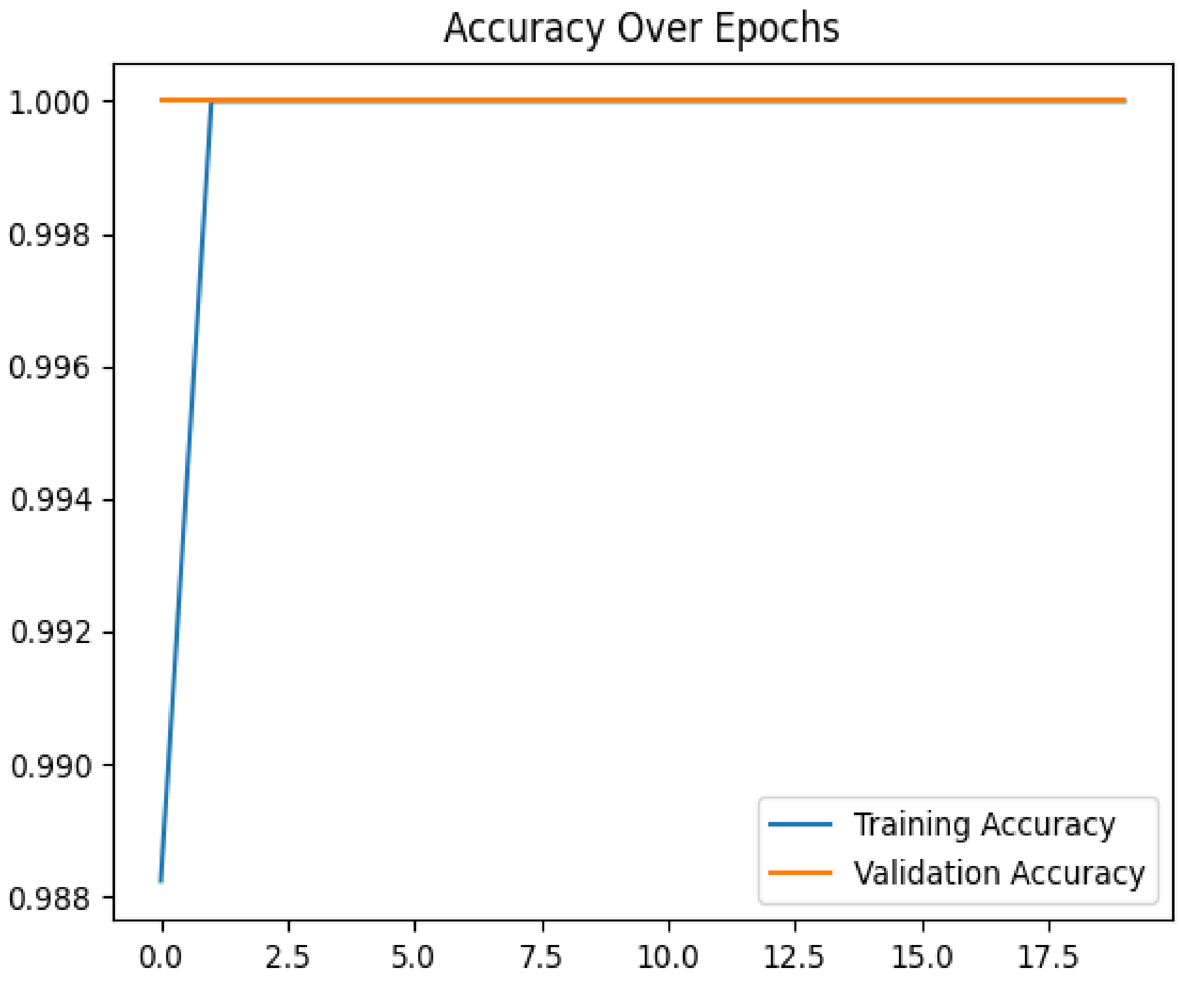


Image 17

  - Training Accuracy: The accuracy of the training set is almost 100% from the first epoch and remains at this level throughout the training process.
  - Validation Accuracy: The accuracy on the test set is also 100% from start to finish.

- Learning Rate Over Epochs: Learning rate (LR) fluctuates and decreases at certain epochs

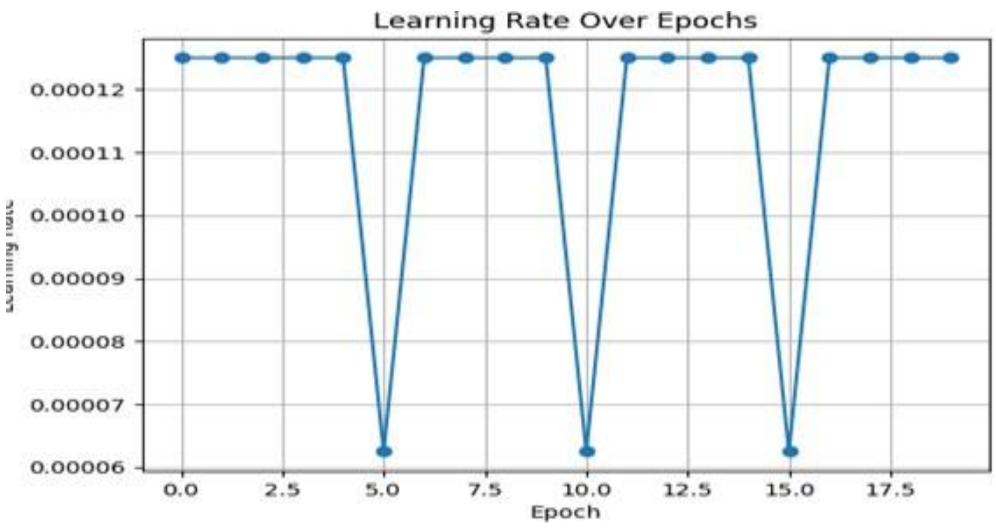


Image 18

- Bias and Variance Visualization:

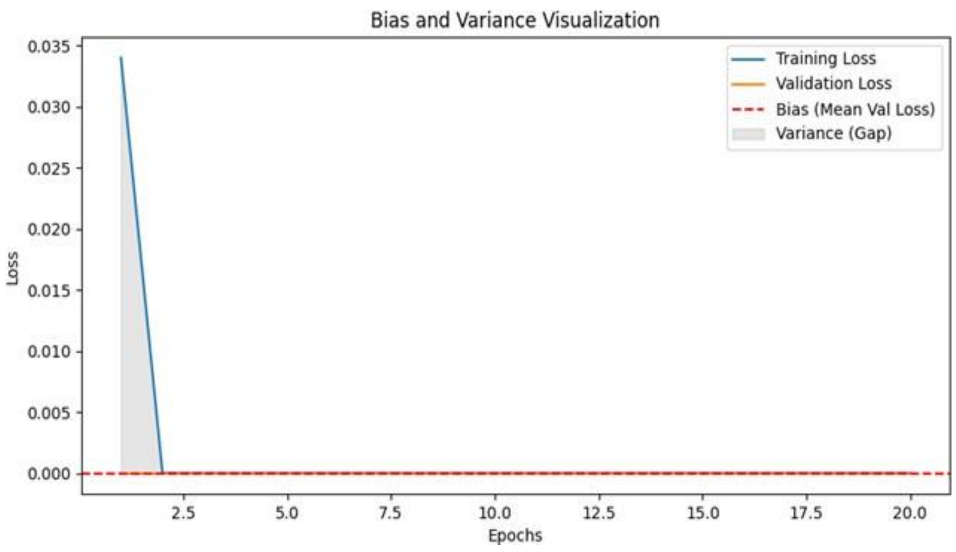


Image 19

    - Training loss drops sharply and quickly reaches a value close to 0 right from the first epochs.
    - Validation loss also drops sharply and reaches a very low value (close to 0) right from the first epoch.
    - From epoch 2 onwards, both training loss and validation loss remain at very low levels and do not change significantly.

CHAPTER V: CONCLUSION AND DEVELOPMENT DIRECTION

5.1. Conclusion

In this research, we successfully developed a machine learning model using Convolutional Neural Networks (CNN) for automatic wildlife identification. The model achieved an accuracy of 100% on the test dataset, demonstrating its ability to accurately classify wild animals from images. This project highlights the potential of machine learning for wildlife monitoring and conservation efforts.

5.2. Limitations

- Limited Dataset: The current model was trained on a relatively small and potentially limited dataset. This can restrict the model's ability to generalize to new and unseen data, particularly for species not well-represented in the training set.

- Overfitting: Despite achieving high accuracy, the model might be overfitting to the training data, especially given the rapid convergence and near-perfect performance on the training set. This could lead to poor performance on real-world data with variations not captured in the training set.

- Limited Species Diversity: The current study focused on a limited number of animal species. Expanding the model to encompass a wider range of species would require significant data collection and model retraining.

- Real-world Applicability: The model was evaluated under controlled conditions. Its performance in real-world scenarios, such as in the field with varying lighting, camera angles, and environmental conditions, may be different.

5.3. Future Development

- Data Expansion: Collect a more diverse dataset by incorporating images from various sources, including camera traps and citizen science initiatives. This will improve the model's generalizability and ability to recognize new species.

- Architectural Refinement: Experiment with advanced CNN architectures such as ResNet, Inception, or EfficientNet to potentially improve accuracy and reduce computational costs.

- Multi-task Learning: Extend the model's capabilities by incorporating additional tasks like object detection and instance segmentation.

- Real-world Deployment: Evaluate the model's performance in real-world scenarios, such as wildlife monitoring systems. Address challenges related to varying lighting conditions, camera angles and environmental factors.

REFERENCES

1. **The dataset used to carry out the project:**

https://docs.ultralytics.com/vi/datasets/detect/african-wildlife/#usage

2. **The theory of topic "The CNN model in image classification":**

https://viblo.asia/p/ung-dung-convolutional-neural-network-trong-bai-toan-phan-loai-anh-4dbZNg8ylYM